INT J COMPUT COMMUN, ISSN 1841-9836
9(2):217-226, April, 2014.

A Metamodel for an Adaptive Control System

F. Valles-Barajas

Fernando Valles-Barajas
Universidad Regiomontana, Information Technology Department
15 de Mayo 567 pte., C.P. 64000 colonia centro, Monterrey,
Nuevo León, México, Tel. +52 81 8220-4733
E-mail: fernando.valles@acm.org, fernando.valles@ieee.org

Abstract: In this paper a metamodel for an Adaptive Control System (ACS) is described. This metamodel was built employing USE, which is a UML-based specification environment. The main goal of the metamodel is to complement other models describing different views of an ACS. As the reader will notice, the metamodel is composed of a graphical and a mathematical model. Weak constraints are specified in the graphical model using a Unified Modeling Language (UML) class diagram, while strong constraints are defined in the mathematical model using the Object Constraint Language (OCL).
Keywords: Adaptive Control, metamodels, OCL, UML

1 Introduction

A software process describes who is doing what, how, and when [9]. One of the phases of a software process is design. In this phase a model of a system, that will later be implemented, is constructed [15]. This model is useful to detect flaws but also for documenting and to establish a communication channel with system's user. Depending on the process being constructed designers can use text or mathematics or a combination of both to build models. An advantage of using text, which is an informal technique, is that the resulting design is easy to understand and can be rapidly constructed. Its disadvantage is that sometimes a model made using an informal technique can lead to a misunderstanding. Formal models, which rely on mathematics, do not have this disadvantage.

On the other hand, control systems are used in industry to assist control engineers in maintaining processes in a desired state (see [3], [25], [10]). Control algorithms are embedded in control systems. Sometimes control systems are applied to control critical systems, whichrequire free errors designs and because of one of the part of control systems is software, recently control engineers are applying software processes in the building of control algorithms [18].

Proposal of the paper: Usually a design covers one view of the system being modeled. According with [7], the design views of system are: structural, procedural and behavior. While making the design's structural view, the designer should document constraints affecting system entities and the relations between them [7]. This documentation can be made using text or a formal language to get a more precise specification. In this paper, the author explains how a semi formal modeling language and a formal language can be used to specify constraints affecting the parts of an adaptive control system as well as constraints affecting the relations between them. The Unified Modeling Language (UML) [5], [14] and the Object Constraint Language (OCL) will be used to specify these constraints.

Previous works: The first step in a software process is to gather requirements system. In [17] a requirements process for control system software is presented. This process is based mainly in

the Rational Unified Process (RUP).
As is recommended in Personal Software Process (PSP) [7], once the requirements are captured by software engineers, the next step is to make a pre-design for the purpose of prediction and planning. In [19] a proposal for the building of pre-design for control systems is presented.
Further details, not covered in the pre-design, can be specified in the design phase. PSP proposes that design can be analyzed from several perspectives. In [20] a proposal, based on PSP, for modeling the structural view of control systems is introduced; in this view the entities of the system, its attributes and relations among the system entities are modeled. Once the entities of system are specified, software designer should detect entities having several states and model them using a state machine. In [21] the authors model an adaptive control system using a state machine.
The design views proposed in PSP are a subset of the UML diagrams, which model systems in more detail. In [23] a survey of the application of UML to model mechatronics systems is presented.
Once software design is made, code must be built. In [18] and in [22] the authors explain best practices in using programming languages at the moment code for control system software is made.

Related works: The Z language is a formal modeling language based on first-order logic and set theory. This modeling language has been applied to the specification of critical systems. In [8], this language is used to specify a control program for a radiation therapy machine.
Another application of the Z modeling language in critical systems is described in [15], where a system that monitors the blood glucose level of diabetics and automatically injects insulin when it is required is specified in the Z language.
The disadvantage of the Z language in comparison with OCL, which is the modeling language used in this paper, is that OCL complements UML and it does not use mathematical symbols to make specifications. The latter characteristic could be attractive to control engineers not familiar with mathematical logic.

Control systems are usually modeled using differential equations and analyzed using, for example, Lyapunov stability theory. In [2] a novel approach based on Hoare logic for reasoning about control systems is presented.
In [12] the authors present intent-specifications model for a robotic software control system. According to the authors, an intent-specification is composed of seven levels that as a whole model the entire software control system. Each level models the system from a different perspective and, in particular, in level 4 a design representation of the system is included.
The Unified Modeling Language (UML) has also been used to model software for control systems. In [24] the authors propose a methodology for generating code for Programmable Logic Controllers (PLCs) from UML diagrams. The application of UML for process control was evaluated from a usability and cognitive science point of view. The authors performed an experiment to evaluate the acceptation of UML for control engineers.
Some researchers are studying mapping between traditional tools used by control engineers for software modeling and UML. For example, in [16] the author analyzed mapping between function blocks, which are defined by the International Electro-technical Commission as the basic construct for distributed control applications and UML. When compared with UML, function blocks do not consider all the benefits of object oriented theory. Another paper including functional blocks and UML is [13]; in that paper the authors apply UML activity diagrams to model function blocks.
Theorem provers have also been used to specify and verify the correctness of control software.

For example, in [6] the authors apply the interactive theorem prover PVS to model the Light Control System, which is a benchmark in formal methods.

Outline of the paper: In this section, the motivation of this work has been explained. Section 2 contains a brief description of an adaptive control system. Section 3 describes the tools used in this paper to specify the constraints of an adaptive control system. In section 4 the adaptive control system design is included. The last section contains concluding remarks.

2 Adaptive Control Systems

Fig. 1 shows the configuration of an indirect adaptive control system. In this kind of adaptive control, a model of the process ($Gp(z^{-1})$) is obtained based on a set of input-output measurements ($u(k), y(k)$) and then the controller ($Gc(z^{-1})$) is designed using this model [10]. The

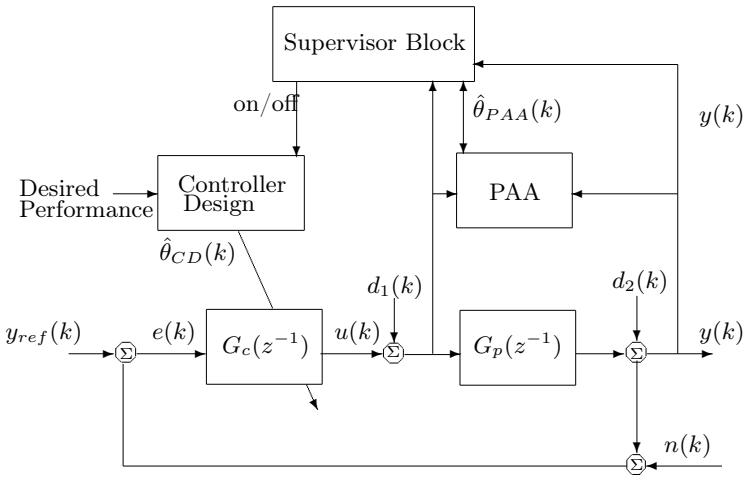

Figure 1: Block diagram for an Adaptive Control System

parameter adaptation algorithm (PAA) block is the responsible for obtaining the parameter vector of the process (see $\hat{\theta}_{PAA}(k)$) in fig. 1). The controller design block specifies the parameters of the controller ($\hat{\theta}_{CD}(k)$) based on the model obtained by the PAA and on the desired performance specified by the system operator.
An adaptive control system must control a process in spite of disturbances $d_1(k)$, $d_2(k)$, noise $n(k)$ and the parametric variations of the process. The supervisor block is in charge of detecting any event that may provoke a decreasing in the performance of system; in these cases the supervisor will turn off the controller G_c
In fig. 1, $y_{ref}(k)$ is the reference, $e(k)$ is the control error, $u(k)$ is the manipulated variable and k is the k^{th} sampling time.

3 UML & OCL: tools to model software systems

UML is a modeling language, based on object oriented theory, developed by the three amigos with the aim of modeling complex software [14]. Every of the UML diagrams models a different view of a system; for example class diagrams specify system entities and the relations between

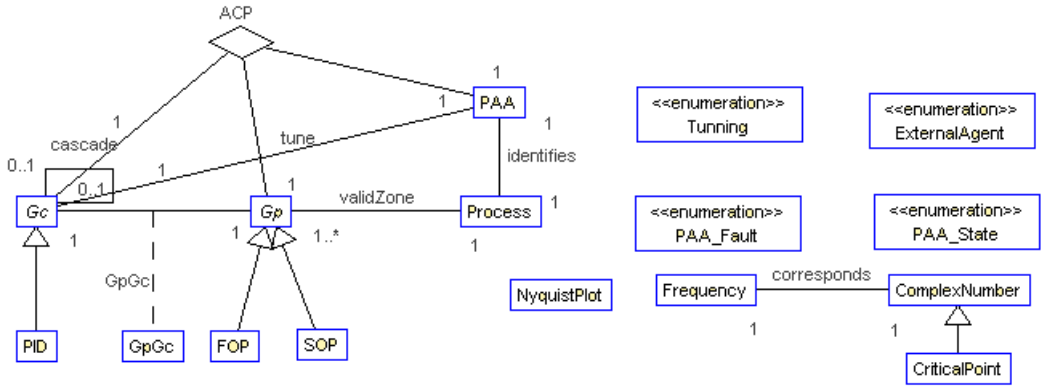

Figure 2: Class diagram for an Adaptive Control System

them. UML has acquired a good acceptation in the software community, though sometimes models created in UML lead to miscommunication.
In these cases it is recommended to complement UML models with an OCL specification, which is a formal modeling language based on logic and set theory. One of the advantages of OCL is that it does not use mathematical symbols to build models; instead it uses a textual representation. For designers without strong background in logic, this characteristic of OCL may result attractive.

4 UML/OCL specification for an Adaptive Control System

In this section the Adaptive Control System metamodel is presented. First of all, a graphical model is described using UML and as will be shown this model contains weak constraints on the elements composing an Adaptive Control System. With the introduction of the UML model the disadvantages of only using graphical models to specify an Adaptive Control System are demonstrated.
A model complementing the graphical model is then constructed using OCL. As the reader will see the weak constraints established in the UML model will be strengthened by using OCL.

4.1 UML specification

Fig. 2 contains a UML class diagram for the ACS of fig. 1. For simplicity, this diagram does not contain all of the details of an ACS; for example, the supervision block and the controller design block are not considered in this diagram. Fig. 3 and 4 contain the USE specification of the class diagram of fig. 2.

As the reader may notice a composition relation between classes G_p (process model), G_c (controller) and PAA (Parameter Adaptation Algorithm) was specified (see the composition ACS in the class diagram of fig. 2 and its definition in the USE specification of fig. 4).
Also, an association class between classes G_p and G_c was defined. As can be seen in the USE specification of fig. 4, this class was defined as an association class because it defines attributes that do not belong to a particular class; for example T_s (time sampling) is associated to the entire system and not to a particular class.
A reflexive relation, named cascade, was defined in the class G_c. Master and slave roles of this relation are specified in fig. 4.

```
model AdaptiveControlSystem

enum ExternalAgent {d1, d2, n}
enum PAA_State {on, off}
enum PAA_Fault {poorExcitation, perturbation, lowM}
enum Tuning {quarterDecayRatio, stepResponse,
  IAE, ITAE, other}

abstract class Gc
attributes
  id : Integer
  isInAutomatic: Boolean
  R: Sequence(Real)
  S: Sequence(Real)
  T: Sequence(Real)
  tuningType: Tuning
  indicator: String
constraints
  inv updateDisplay:
    isInAutomatic implies indicator = 'automatic'
  inv ValidId:
    id >= 1
end

class PAA
attributes
  theta: Set(Real)
  phi: Set(Real)
  lambda: Real
  state: PAA_State
  faults: Set(PAA_Fault)
operations
  updateGp( B: Sequence(Real), A: Sequence(Real) )
constraints
 inv:
  if faults->notEmpty then state = #off
  else state = #on
  endif
end

class PID < Gc
attributes
  tao_d: Real
  tao_i: Real
  Kc: Real
operations
  assignGp(gp: Gp)
end

abstract class Gp
attributes
  B: Sequence(Real)
  A: Sequence(Real)
  d: Integer
  na: Integer
  nb: Integer
  np: NyquistPlot
operations
  isMonicHurwitzPolynomial (): Boolean
constraints
  inv sructure3:
     d>0 and na>0 and nb>0
  inv structure:
    na >= nb
  inv structure2:
    A->size() = na and B->size() = nb
end

class FOP < Gp
attributes
  K: Real
  theta: Real
  tao: Real
operations
  initGpFOP()
  fromContinousToDiscrete()
constraints
  inv structure:
    nb = 1 and na = 2
end

class SOP < Gp
constraints
  inv structure:
    nb = 2 and na = 3
end
```

Figure 3: USE specification for an Adaptive Control System (part I)

```
class Process
attributes
  identifiedGp: Gp
  affected: Set(ExternalAgent)
end

class Frequency
attributes
  value: Real
end

associationclass GpGc between
  Gp [1] role model2;
  Gc [1] role controller;
attributes
  Ts: Integer
  affectedBy: Set(ExternalAgent)
  faultHistory: Bag(ExternalAgent)
  gainMargin: Real
  phaseMargin: Real
  modulusMargin: Real
  delayMargin: Real
operations
  registerAFault(f : ExternalAgent)
end

association tune between
  PAA [1] role tunner;
  Gc [1] role controller2;
end

association corresponds between
  Frequency [1]
  ComplexNumber [1]
end

class ComplexNumber
end

class CriticalPoint < ComplexNumber
end

class NyquistPlot
attributes
  w: Sequence(Frequency)
  c: Set(ComplexNumber)
end

association cascade between
  Gc [0..1] role master;
  Gc [0..1] role slave;
end

association validZone between
  Process [1] role plant;
  Gp [1..*] role Model;
end

association identifies between
  PAA [1] role identifier;
  Process [1] role process;
end

composition ACP between
  PAA [1]
  Gp [1]
  Gc [1]
end
```

Figure 4: USE specification for an Adaptive Control System (part II)

The model defined in fig. 2, 3 and 4 contain some flaws; for example this model allows that a controller G_{c_1} plays at the same time master and slave role in the cascade association.
Another design flaw occurs between classes G_c and G_p as is explained as follows. PID controllers have demonstrated to have a good performance when the process being controlled can be modeled as a first or second order process (see [11]), however the class diagram of fig. 2 allows that any kind of process may participate in the relation between G_c and G_p.
These design flaws justify the addition of OCL constraints to the ACS class diagram model shown in fig. 2 and defined in fig. 3 and 4.

4.2 OCL specification

In this section, invariants on the class diagram defined in fig. 2 are defined. Some invariants affect more than two elements of the class diagram. Invariants are explained in the same order they appear in fig. 5 and 6.

Fig. 5 defines three constraints on the class controller, G_c. Invariant *onlyOneRole* assures that a controller G_{c_1} cannot be at the same time, both a master controller and a slave controller. Invariant *differentIDs* constraints that in a cascade relation the controller playing the master role has a different *id* that the controller playing the slave role.
The last invariant of the class G_c specifies that if the type of controller is a PID controller, then an appropriate tuning type must be selected (quarterDecayRatio, stepResponse, IAE or ITAE

```
context Gc
  inv onlyOneRole:
    master->notEmpty implies slave->isEmpty

 inv differentIDs :
  (master->notEmpty implies self.id <> master.id) and (slave->notEmpty implies self.id <> slave.id)

  inv validGp:
    self.oclIsTypeOf( PID ) implies
      tuningType <> #other and (model2.oclIsTypeOf( FOP ) or model2.oclIsTypeOf( SOP ))
          and model2.oclIsTypeOf( FOP ) implies model2.oclAsType( FOP ).theta < gpGc.Ts

context PID::assignGp( gp: Gp )
  pre: FOP.allInstances->union( SOP.allInstances )->includes( gp )
  post: model2.plant.identifiedGp = gp

context NyquistPlot
  inv orderedFreq1:
    let integers = Sequence{1 .. w->size()} in
      integers->forAll(e1, e2: Integer | w->at(e2).value > w->at(e1).value implies e2 > e1)

  inv orderedFreq2: -- another way to specify invariant orderedFreq1
    w->forAll(f1, f2: Frequency | w->indexOf(f2) > w->indexOf(f1) implies f2.value > f1.value)

  inv stability :
    w->forAll(f: Frequency | CriticalPoint.allInstances->excludes(f.complexNumber))

  inv completeness:
    w->forAll(f: Frequency | ComplexNumber.allInstances->exists(cn | f.complexNumber=(cn)))
```

Figure 5: USE constraints for an Adaptive Control System (part I)

among others, see the enumeration Tuning defined in fig. 3). Besides this, the process being controlled should be modeled as a first order process or as a second order process. If the process is a first order process then the dead time, θ, should be smaller than T_s.

A method to assign a process model to a PID controller is defined in fig. 5. This method forces that the model, G_p, should be either a first order process or a second order process.

Four invariants are defined on the class NyquistPlot. Invariants *orderedFreq1* and *orderedFreq2* declares that the frequencies specified in a Nyquist plot should be ordered from low to high.
Invariant *stability* defines that a Nyquist plot should not contain the critical point.
The last invariant of class NyquistPlot specifies that every frequency in a Nyquist plot has a complex number assigned to it.

Fig. 6 declares a method of the class G_p, which constraints that the polynomial A of G_p should be monic; in other words the leading coefficient should be 1.

The method *updateGp* defined on the class PAA, updates the model G_p as long as the model identified by the PAA is inside the valid zone of the process being controlled and there is not fault affecting the entire system.

Invariant *stabilityCriteria* defines the standard values to get a robust controller recommended in [3]. Method *registerAFault* defined on the association class *GpGc* updates the logging of system faults.

```
context Gp :: isMonicHurwitzPolynomial(): Boolean
  pre: A->first() = 1.0
  post: result = true

context PAA::updateGp( B: Sequence(Real), A: Sequence(Real) )
  pre: state = #on
  pre: process.Model->exists(gp: Gp | gp.A = A and gp.B = B)
  pre: Set{faults, process.Model.gpGc.affectedBy}->isEmpty

  post: process.identifiedGp.A = A and process.identifiedGp.B = B

context GpGc
  inv stabilityCriteria :
    gainMargin >= 6 and modulusMargin >= -6
      and Sequence{ 30, 31 .. 59, 60 }->includes(phaseMargin) and delayMargin = 0.1*Ts

context GpGc::registerAFault(f : ExternalAgent)
  post: faultHistory = faultHistory@pre->including(f)

context Process
  inv not_a_PID:
   let ip = identifiedGp in let fop = ip.oclAsType( FOP ) in
     ip.oclIsTypeOf( FOP ) and (fop.theta > fop.tao*0.25 and fop.theta < ip.gpGc.Ts) implies
         PID.allInstances->excludes( ip.controller )
```

Figure 6: USE constraints for an Adaptive Control System (part II)

The last invariant of fig. 6 constrains the class Process. If the process is modeled as a first order process then θ should be less than 0.25τ (τ is the constant time of a first order process) and less than T_s.

5 Conclusions

In this paper an informal modeling language (UML) and a formal modeling language (OCL) have been applied to model an adaptive control system. UML was used to specify weak constraints whereas OCL was applied to strength the model made in UML.
The author of this paper believes that the model included in this work, will be useful for control engineers that want to have a better understanding of how to apply UML and OCL in the modeling of control systems.
A simple example (an Adaptive Control System) was used in this paper to illustrate how control system software can be specified using an informal notation (UML) together with a formal modeling language (OCL). The technique proposed in this paper should be used to model software for complex control systems, e.g. nuclear plants, robots, planes among others.

The future lines of research based on this paper are:

- Due to the fact that UML is a general purpose modeling language, in some cases a more specialized language gives a more precise specification than that obtained using UML. This drawback of UML has been studied and analyzed by the UML community and to overcome this problem UML extension mechanisms have been proposed; one of these are profiles. A profile based on UML and OCL will be studied in future papers.

- There are some tools to prove if some constraints are fulfilled by one specification (see for example [1] and [4], among others). Building one tool with this characteristic would be useful to check the models created by the technique proposed in this paper.

- Every modeling technique specifies a system from a particular perspective. In this paper, a UML class diagram was used to model the entities composing a control system along with the constraints among them; strong constraints were specified using OCL. Other perspectives not considered in this paper could model the interaction between the entities composing the system and the interaction between the user of a system and the system itself. A study of other models complementing the perspectives of control system software will be made in a future research.

Bibliography

[1] R. Arthan, P. Caseley, C. O'Halloran and A. Smith (2000). *ClawZ: Control laws in Z*, Third IEEE International Conference on Formal Engineering Methods (ICFEM 2000), 2000.

[2] R. Arthan, U. Martin, E. A. Mathiesen and P. Oliva (2007). Reasoning about linear systems, *Proceedings of 5th IEEE International Conference on Software Engineering and Formal Methods SEFM*, 2007.

[3] K. J. Åström and R. M. Murray (2012). *Feedback Systems: An Introduction for Scientists and Engineers, Version v2.11b*, Princeton University Press.

[4] S. Bensalem, P.Caspi, C.Parent-Vigouroux and C.Dumas (1999). A methodology for proving control systems with Lustre and PVS, 7^{th} *Working Conference on Dependable Computing for Critical Applications (DCCA7)*, San Jose, January 1999.

[5] M. Fowler (2003). *UML Distilled*, Addison-Wesley, 3^{rd} edition, 2003.

[6] A. de Groot, J. Hooman (20000), Analyzing the Light Control System with PV, *Journal of Universal Computer Science*, vol. 6, no. 7.

[7] W. S. Humphrey (2005). *PSP : A Self-Improvement Process for Software Engineers*, Addison-Wesley.

[8] J. Jacky, J. Unger, M. Patrick, D. Reid and R. Risler (1997). Experience with Z developing a control program for a radiation therapy machine, *Lecture Notes in Computer Science*, 1212:317-328.

[9] P. Kruchten (2003). *The Rational Unified Process: An Introduction*, Addison-Wesley Professional, 3^{rd} edition.

[10] I. D. Landau and G. Zito (2006). *Digital Control Systems: Design, Identification and Implementation*, Springer.

[11] A. Leva, C. Cox, and A. Ruano (2002). *Hands-on PID autotuning: a guide to better utilization*, IFAC Professional Briefs, 2002.

[12] I. Navarro, K. Lundqvist, and N. Leveson (2001) An intent-specifications model for a robotic software control system, 20^{th} *Conference Digital Avionics Systems*, 2001. DASC.

[13] S. Panjaitan, G. Frey (2005), Functional Design for IEC 61499 Distributed Control Systems using UML Activity Diagrams. *Proceedings ICICI 2005*, Bandung, Indonesia, 64-70.

[14] J. Rumbaugh, I. Jacobson, and G. Booch (2004). *The Unified Modeling Language Reference Manual*, Addison-Wesley Professional, 2^{nd} edition.

[15] I. Sommerville (2010). *Software Engineering*, Addison-Wesley, 9^{th} edition.

[16] K. C. Thramboulidis 92004), Using UML in Control and Automation: A Model Driven Approach, 2^{nd} *IEEE International Conference on Industrial Informatics INDIN'04*, 24th -26th June, 2004, Berlin, Germany.

[17] F. Valles-Barajas (2007), A requirements engineering process for control engineering software, *Innovations in Systems and Software Engineering: A NASA Journal*, 3(4):217-227.

[18] F. Valles-Barajas and W. Schaufelberger (20080, Use of object-oriented languages in control engineering, *Journal of Research in Computing Science*, vol.36. 2008.

[19] F. Valles-Barajas (2009), A SysML requirements model for the 1992 ACC robust control benchmark, *Information Technology and Control*, 38(3):245-251. 2009.

[20] F. Valles-Barajas and W. Schaufelberger (2010), A proposal for the software design of control systems based on the personal software process, *International Journal of Innovative Computing, Information and Control*, 6(8):3451-3466. 2010.

[21] F. Valles-Barajas (2010), A Novel Model for Adaptive Control Systems; A State Machine Approach, *Int J Comput Commun*,ISSN 1841-9836, 5(3):292-300. 2010.

[22] F. Valles-Barajas (2011), A survey of high-level programming languages in control systems, *The International Arab Journal of Information Technology*, 8(2):178-187.

[23] F. Valles-Barajas (2011), A survey of UML applications in mechatronic systems, *Innovations in Systems and Software Engineering: A NASA Journal*, 7(1):43-51.

[24] B. Vogel-Heuser, D. Friedrich, U. Katzke and D. Witsch (2005), Usability and benefits of UML for plant automation some research results *Engineering*, vol. 3, no. 1.

[25] B. Wittenmark, K. J. Åström, and K. E. Årzén. Computer control: An overview, *IFAC Professional Briefs*, 2002.